ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ № 3

з курсу «ООП»

студентки групи ПС-21-1

Байдачної Ірини Вікторівни

кафедра комп’ютерних технологій,

ДНУ 2022/2023 н.р.

1. **Постановка задачі.**

Скласти об’єктно-орієнтовану програму на мові C++, яка в діалоговому режимі керує графічними

об’єктами, що відображаються на екрані дисплею.

По натисненню клавіши F1 програма виводить на екран коротку підказку по усіх наявних

командах/клавішах; наприклад: як створити об’єкт, як зрушити з місця, як перейти до «наступного» об’єкту

тощо.

1. **Опис розв’язку.**

Для функціонування программи було створено такі класи:

Composite.h – для створення композиту;

CompositeMemento.h – для запам’ятовування стану композиту; CompositeUtils.h – для функцій, які роблять запам’ятовування композиту;

Figure.h – базовий класс фігури;

Memento.h - для запам’ятовування стану композиту;

Rectangle.h – клас прямокутника;

RectangleMemento.h – для запам’ятовування стану прямокутника;

RectangleUtils.h – для функцій, які роблять запам’ятовування стану прямокутника;

SceneController.h – для керування сценою;

Square.h – клас квадрату;

SquareMemento.h – для функцій, які роблять запам’ятовування стану квадрату;

SquareUtils.h – для функцій, які роблять запам’ятовування квадрату;

1. Вихідний текст програми розв’язку задачі (основні фрагменти з коментарями).

**Composite.cpp:**

#include "Composite.h"

void Composite::add(Figure\* fig)

{

if (fig->get\_figure\_type() == "composite") {

Composite\* com = (Composite\*)fig;

com->set\_position(x, y);

composites.push\_back(com);

}

else {

composites.push\_back(fig);

}

}

int Composite::get\_x()

{

return x;

}

int Composite::get\_y()

{

return y;

}

void Composite::draw(RenderWindow& window)

{

for (int i = 0; i < composites.size(); i++) composites[i]->draw(window);

}

void Composite::move(Vector2u window\_size, float coff, int speed)

{

for (int i = 0; i < composites.size(); i++) {

composites[i]->move(window\_size, coff, speed);

}

}

void Composite::limit\_move(Vector2u size)

{

for (int i = 0; i < composites.size(); i++) composites[i]->limit\_move(size);

}

FloatRect Composite::get\_bounds()

{

return composites[0]->get\_bounds();

}

void Composite::set\_position(int x, int y)

{

for (int i = 0; i < composites.size(); i++) composites[i]->set\_position(x, y);

}

void Composite::set\_color(Color color)

{

color = color;

for (int i = 0; i < composites.size(); i++) composites[i]->set\_color(color);

}

void Composite::set\_fill\_color(Color color)

{

for (int i = 0; i < composites.size(); i++) composites[i]->set\_fill\_color(color);

}

void Composite::hide()

{

switch (hide\_flag)

{

case 1:

for (int i = 0; i < composites.size(); i++) {

composites[i]->set\_fill\_color(Color::Black);

}

hide\_flag++;

break;

case 2:

for (int i = 0; i < composites.size(); i++) {

composites[i]->set\_fill\_color(composites[i]->get\_color());

}

hide\_flag--;

break;

}

}

void Composite::transform()

{

switch (transform\_flag) {

case 1:

for (int i = 0; i < composites.size(); i++) {

composites[i]->transform();

}

transform\_flag++;

break;

case 2:

for (int i = 0; i < composites.size(); i++) {

composites[i]->transform();

}

transform\_flag--;

break;

}

}

void Composite::remove(RenderWindow\* window)

{

for (int i = 0; i < composites.size(); i++) {

composites[i]->remove(window);

}

}

Composite\* Composite::clone()

{

Composite\* comp = new Composite();

for (int i = 0; i < composites.size(); i++) {

Figure\* new\_figure = nullptr;

if (composites[i]->get\_figure\_type() == "square") {

Square\* sq = (Square\*)composites[i];

new\_figure = new Square(sq->get\_width(), sq->get\_color(), sq->get\_x(), sq->get\_y());

}

else if (composites[i]->get\_figure\_type() == "rectangle") {

Rectangle\* rec = (Rectangle\*)composites[i];

new\_figure = new Rectangle(rec->get\_width(), rec->get\_height(), rec->get\_color(), rec->get\_x(), rec->get\_y());

}

else if (composites[i]->get\_figure\_type() == "composite") {

Composite\* cmp = (Composite\*)composites[i];

new\_figure = cmp->clone();

}

comp->add(new\_figure);

}

return comp;

}

void Composite::set\_memento(CompositeMemento& memento)

{

x = memento.basic.x;

y = memento.basic.y;

composites = memento.figures;

set\_position(x, y);

}

void Composite::get\_memento(CompositeMemento& memento)

{

memento.basic.x = x;

memento.basic.y = y;

memento.figures = composites;

}

**CompositeMemento.cpp:**

#include "CompositeMemento.h"

ostream& operator<<(ostream& out, const CompositeMemento& memento)

{

out << memento.basic;

for (int i = 0; i < memento.figures.size(); i++) {

if (memento.figures[i]->get\_figure\_type() == "square") {

SquareMemento mem;

Square\* fig = (Square\*)memento.figures[i];

fig->get\_memento(mem);

out << mem;

}

else if (memento.figures[i]->get\_figure\_type() == "rectangle") {

RectangleMemento mem;

Rectangle\* rect = (Rectangle\*)memento.figures[i];

rect->get\_memento(mem);

out << mem;

}

}

return out;

}

istream& operator>>(istream& in, CompositeMemento& memento)

{

in >> memento.basic;

for (int i = 0; i < memento.figures.size(); i++) {

if (memento.figures[i]->get\_figure\_type() == "square") {

SquareMemento mem;

Square\* fig = (Square\*)memento.figures[i];

in >> mem;

fig->set\_memento(mem);

}

else if (memento.figures[i]->get\_figure\_type() == "rectangle") {

RectangleMemento mem;

Rectangle\* rect = (Rectangle\*)memento.figures[i];

in >> mem;

rect->set\_memento(mem);

}

}

return in;

}

**CompositeUtils.cpp:**

#include "CompositeUtils.h"

void CompositeUtils::save\_state(Composite\* composite, int counter)

{

ifstream in(filename);

in.seekg(counter, ios::beg);

in >> memento;

composite->set\_memento(memento);

in.close();

}

int CompositeUtils::load\_state(Composite\* composite, int before)

{

composite->get\_memento(memento);

ofstream out(filename, ios::app);

out << memento;

int after = out.tellp();

int numberOfBytesWritten = after - before;

composite->set\_bytes\_written(numberOfBytesWritten);

out.close();

return after;

}

**Figure.cpp:**

#include "Figure.h"

**lab4.cpp:**

#include "SceneController.h"

#include <Windows.h>

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "ukr");

SetConsoleOutputCP(1251);

srand(time(NULL));

SceneController\* controller = SceneController::get\_instance(argv[1]);

controller->start();

}

**Memento.cpp:**

#include "Memento.h"

ostream& operator<<(ostream& out, const Memento& memento)

{

out << memento.color.r << "\n";

out << memento.color.g << "\n";

out << memento.color.b << "\n";

out << memento.x << "\n";

out << memento.y << "\n";

return out;

}

istream& operator>>(istream& in, Memento& memento)

{

in >> memento.color.r;

in >> memento.color.g;

in >> memento.color.b;

in >> memento.x;

in >> memento.y;

return in;

}

**Rectangle.cpp:**

#include "Rectangle.h"

int Rectangle::get\_x()

{

return x;

}

int Rectangle::get\_y()

{

return y;

}

void Rectangle::draw(RenderWindow& window)

{

window.draw(shape);

}

void Rectangle::move(Vector2u window\_size, float coff, int speed)

{

shape.move(speed \* coff \* 3.0f, 0.0f);

x = shape.getPosition().x;

y = shape.getPosition().y;

limit\_move(window\_size);

}

void Rectangle::limit\_move(Vector2u size)

{

if (shape.getPosition().x >= size.x) set\_position(0, shape.getPosition().y);

else if (shape.getPosition().x <= 0) set\_position(size.x, shape.getPosition().y);

if (shape.getPosition().y >= size.y) set\_position(shape.getPosition().x, 0);

else if (shape.getPosition().y <= 0) set\_position(shape.getPosition().x, size.y);

}

FloatRect Rectangle::get\_bounds()

{

return shape.getGlobalBounds();

}

void Rectangle::set\_position(int x, int y)

{

this->x = x;

this->y = y;

shape.setPosition(x, y);

}

void Rectangle::set\_color(Color color)

{

this->color = color;

shape.setFillColor(color);

}

void Rectangle::set\_fill\_color(Color color)

{

shape.setFillColor(color);

}

void Rectangle::hide()

{

switch (hide\_flag) {

case 1:

shape.setFillColor(Color::Black);

hide\_flag++;

break;

case 2:

shape.setFillColor(color);

hide\_flag--;

break;

}

}

void Rectangle::transform()

{

switch (transform\_flag) {

case 1:

width -= 30;

height -= 30;

transform\_flag++;

break;

case 2:

width += 30;

height += 30;

transform\_flag--;

break;

}

shape.setSize(Vector2f(width, height));

}

void Rectangle::remove(RenderWindow\* window)

{

window->clear();

}

Rectangle\* Rectangle::clone()

{

Rectangle\* rect = new Rectangle(width, height, color, x, y);

return rect;

}

void Rectangle::set\_memento(RectangleMemento& memento)

{

color = memento.basic.color;

height = memento.height;

width = memento.width;

x = memento.basic.x;

y = memento.basic.y;

shape.setPosition(x, y);

shape.setFillColor(get\_color());

shape.setSize(Vector2f(width, height));

}

void Rectangle::get\_memento(RectangleMemento& memento)

{

memento.basic.color = color;

memento.height = height;

memento.width = width;

memento.basic.x = x;

memento.basic.y = y;

}

**RectangleMemento.cpp:**

#include "RectangleMemento.h"

ostream& operator<<(ostream& out, const RectangleMemento& memento)

{

out << memento.basic << "\n";

out << memento.width << "\n";

out << memento.height << "\n";

return out;

}

istream& operator>>(istream& in, RectangleMemento& memento)

{

in >> memento.basic;

in >> memento.width;

in >> memento.height;

return in;

}

**RectangleUtils.cpp:**

#include "RectangleUtils.h"

void RectangleUtils::save\_state(Rectangle\* rectangle, int counter)

{

ifstream in(filename);

in.seekg(counter, ios::beg);

in >> memento;

rectangle->set\_memento(memento);

in.close();

}

int RectangleUtils::load\_state(Rectangle\* rectangle, int before)

{

rectangle->get\_memento(memento);

ofstream out(filename, ios::app);

out << memento;

int after = out.tellp();

int numberOfBytesWritten = after - before;

rectangle->set\_bytes\_written(numberOfBytesWritten);

out.close();

return after;

}

**SceneController.cpp:**

**#include "SceneController.h"**

**SceneController\* SceneController::instance;**

**SceneController\* SceneController::get\_instance(string \_fileName)**

**{**

**if (instance == nullptr) instance = new SceneController(\_fileName);**

**return instance;**

**}**

**void SceneController::menu()**

**{**

**cout << "1: рух фігури" << endl;**

**cout << "2: додати фігуру" << endl;**

**cout << "3: видалити фігуру" << endl;**

**cout << "4: клонувати фігуру" << endl;**

**cout << "5: додати до композиту" << endl;**

**cout << "6: змінити колір" << endl;**

**cout << "7: деформація фігури" << endl;**

**cout << "8: збереження до файлу" << endl;**

**cout << "9: завантаження з файлу" << endl;**

**cout << "ЛКМ: обрати фігуру" << endl;**

**cout << "ESC: вихід з програми" << endl;**

**}**

**void SceneController::start()**

**{**

**ofstream out(fileName);**

**out << "";**

**out.close();**

**float move\_coff = 0.1;**

**int coll\_idx = -1;**

**Clock\* keyboard\_clock = new Clock;**

**bool is\_comp\_in\_vector = false;**

**int counter = 0, figures\_counter = 0;**

**Rectangle\* rect = new Rectangle(200, 100, Color::Cyan, 100, 100);**

**Square\* square = new Square(100, Color::Yellow, 500, 600);**

**add\_figure(rect);**

**add\_figure(square);**

**cout << "Натисніть F1 щоб перейти до меню" << endl;**

**bool f1IsPressed = false;**

**//menu();**

**Color main\_color = Color(0, 0, 0);**

**while (window->isOpen()) {**

**if (Keyboard::isKeyPressed(Keyboard::F1) && f1IsPressed == false)**

**{**

**menu();**

**f1IsPressed = true;**

**}**

**window->clear(main\_color);**

**for (int i = 0; i < figures.size(); i++) {**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**Event event;**

**while (window->pollEvent(event)) {**

**if (event.type == Event::Closed) {**

**window->close();**

**}**

**if (Mouse::isButtonPressed(Mouse::Left)) {**

**Vector2i mouse\_pos = Mouse::getPosition(\*window);**

**Vector2f translated\_pos = window->mapPixelToCoords(mouse\_pos);**

**for (int i = 0; i < figures.size(); i++)**

**{**

**if (figures[i]->get\_bounds().contains(translated\_pos)) {**

**selected\_figure = i;**

**figures[i]->set\_selected(true);**

**}**

**}**

**}**

**if (figures[selected\_figure] != nullptr) {**

**if (Keyboard::isKeyPressed(Keyboard::Num1)) {**

**if (figures[selected\_figure]->get\_selected()) {**

**figures[selected\_figure]->move(window->getSize(), move\_coff, 1000);**

**window->clear(main\_color);**

**if (collision\_check()) {**

**figures[selected\_figure]->set\_color(Color::Magenta);**

**}**

**for (int i = 0; i < figures.size(); i++)**

**{**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**else {**

**cout << "Вы должны выбрать фигуру!" << endl;**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**break;**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num3)) {**

**if (figures[selected\_figure]->get\_selected())**

**{**

**if (figures.size() > 1)**

**{**

**figures[selected\_figure]->remove(window);**

**figures.erase(figures.begin() + selected\_figure);**

**selected\_figure--;**

**if (selected\_figure < 0)**

**{**

**selected\_figure = figures.size() - 1;**

**}**

**for (int i = 0; i < figures.size(); i++) {**

**if (figures[i]->get\_figure\_type() == "composite") counter++;**

**}**

**if (counter == 0) {**

**is\_comp\_in\_vector = false;**

**}**

**counter = 0;**

**}**

**else {**

**cout << "Повинен бути хоча б один елемент" << endl;**

**}**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**else {**

**cout << "Оберіть фігуру!" << endl;**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**break;**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num4)) {**

**if (figures[selected\_figure]->get\_selected())**

**{**

**add\_figure(figures[selected\_figure]->clone());**

**window->clear(main\_color);**

**for (int i = 0; i < figures.size(); i++)**

**{**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**else {**

**cout << "Оберіть фігуру!" << endl;**

**break;**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num5)) {**

**if (figures[selected\_figure]->get\_selected()) {**

**Composite\* comp = nullptr;**

**if (!is\_comp\_in\_vector) {**

**comp = new Composite(500, 500);**

**figures.push\_back(comp);**

**is\_comp\_in\_vector = true;**

**}**

**else {**

**cout << "Оберіть композит" << endl;**

**cout << "-1 - новий" << endl;**

**int choice;**

**for (int i = 0; i < figures.size(); i++) {**

**if (figures[i]->get\_figure\_type() == "composite") {**

**comp = (Composite\*)figures[i];**

**cout << i << " (" << comp->get\_count() << " фігур)" << endl;**

**}**

**}**

**cin >> choice;**

**if (choice == -1) {**

**comp = new Composite(500, 200);**

**figures.push\_back(comp);**

**}**

**else if (choice < figures.size() && figures[choice]->get\_figure\_type() == "composite") {**

**comp = (Composite\*)figures[choice];**

**}**

**}**

**Figure\* fig = figures[selected\_figure]->clone();**

**fig->set\_position(comp->get\_x(), comp->get\_y());**

**comp->add(fig);**

**figures[selected\_figure]->remove(window);**

**figures.erase(figures.begin() + selected\_figure);**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**else {**

**cout << "Оберіть фігуру!" << endl;**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**break;**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num6)) {**

**if (figures[selected\_figure]->get\_selected()) {**

**int choice;**

**cout << "1 - blue, 2 - yellow, 3 - red, 4 - green, 5 - magenta" << endl;**

**cout << "Оберіть колір: " << endl;**

**cin >> choice;**

**Color c = colors[choice - 1];**

**figures[selected\_figure]->set\_color(c);**

**window->clear(main\_color);**

**for (int i = 0; i < figures.size(); i++)**

**{**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**else {**

**cout << "Ви повинні обрати фігуру!" << endl;**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**break;**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num7)) {**

**if (figures[selected\_figure]->get\_selected()) {**

**figures[selected\_figure]->transform();**

**window->clear(main\_color);**

**for (int i = 0; i < figures.size(); i++)**

**{**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**else {**

**cout << "Оберіть фігуру!" << endl;**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**break;**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num8)) {**

**int before = 0;**

**figures\_counter = figures.size();**

**for (int i = 0; i < figures.size(); i++) {**

**if (figures[i]->get\_figure\_type() == "rectangle") {**

**RectangleUtils utils;**

**before = utils.load\_state((Rectangle\*)figures[i], before);**

**}**

**else if (figures[i]->get\_figure\_type() == "square") {**

**SquareUtils utils;**

**before = utils.load\_state((Square\*)figures[i], before);**

**}**

**else if (figures[i]->get\_figure\_type() == "composite") {**

**CompositeUtils utils;**

**before = utils.load\_state((Composite\*)figures[i], before);**

**}**

**}**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num9)) {**

**figures.erase(figures.begin() + figures\_counter, figures.begin() + figures.size());**

**counter = 0;**

**for (int i = 0; i < figures.size(); i++) {**

**if (figures[i]->get\_figure\_type() == "rectangle") {**

**RectangleUtils utils;**

**utils.save\_state((Rectangle\*)figures[i], counter);**

**}**

**else if (figures[i]->get\_figure\_type() == "square") {**

**SquareUtils utils;**

**utils.save\_state((Square\*)figures[i], counter);**

**}**

**else if (figures[i]->get\_figure\_type() == "composite") {**

**CompositeUtils utils;**

**utils.save\_state((Composite\*)figures[i], counter);**

**Composite\* comp = (Composite\*)figures[i];**

**if (comp->get\_count() == 0) {**

**figures[i]->remove(window);**

**figures.erase(figures.begin() + i);**

**}**

**}**

**if (i < figures.size()) {**

**counter += figures[i]->get\_bytes\_written();**

**}**

**}**

**if (figures.size() > 0) {**

**selected\_figure = 0;**

**}**

**window->clear(main\_color);**

**for (int i = 0; i < figures.size(); i++)**

**{**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Num2)) {**

**Square\* square;**

**Rectangle\* rect;**

**int width, height, x = 0, y = 0, color, shape;**

**cout << "Введіть координати: ";**

**cin >> x >> y;**

**cout << "Введіть тип фігури: " << endl;**

**cout << "1 - прямокутник" << endl;**

**cout << "2 - квадрат" << endl;**

**cin >> shape;**

**cout << "Введіть колір: " << endl;**

**cout << "1 - синій, 2 - желтый, 3 - червоний, 4 - зеленый, 5 - блакитний" << endl;**

**cin >> color;**

**cout << "Введіть ширину: ";**

**cin >> width;**

**switch (shape) {**

**case 1:**

**cout << "Введіть висоту: ";**

**cin >> height;**

**rect = new Rectangle(width, height, colors[color - 1], x, y);**

**add\_figure(rect);**

**break;**

**case 2:**

**square = new Square(width, colors[color - 1], x, y);**

**add\_figure(square);**

**break;**

**}**

**for (int i = 0; i < figures.size(); i++)**

**{**

**figures[i]->draw(\*window);**

**}**

**window->display();**

**keyboard\_clock->restart();**

**while (keyboard\_clock->getElapsedTime().asSeconds() < 1) {}**

**}**

**if (Keyboard::isKeyPressed(Keyboard::Escape)) { exit(0); }**

**}**

**}**

**}**

**Square.cpp:**

#include "Square.h"

FloatRect Square::get\_bounds()

{

return shape.getGlobalBounds();

}

void Square::set\_position(int x, int y)

{

this->x = x;

this->y = y;

shape.setPosition(x, y);

}

void Square::set\_color(Color color)

{

this->color = color;

shape.setFillColor(color);

}

void Square::set\_fill\_color(Color color)

{

shape.setFillColor(color);

}

int Square::get\_x()

{

return x;

}

int Square::get\_y()

{

return y;

}

void Square::draw(RenderWindow& window)

{

window.draw(shape);

}

void Square::move(Vector2u window\_size, float move\_coff, int speed)

{

shape.move(speed \* move\_coff \* 3.0f, 0.0f);

x = shape.getPosition().x;

y = shape.getPosition().y;

limit\_move(window\_size);

}

void Square::limit\_move(Vector2u size)

{

if (shape.getPosition().x >= size.x) set\_position(0, shape.getPosition().y);

else if (shape.getPosition().x <= 0) set\_position(size.x, shape.getPosition().y);

if (shape.getPosition().y >= size.y) set\_position(shape.getPosition().x, 0);

else if (shape.getPosition().y <= 0) set\_position(shape.getPosition().x, size.y);

}

void Square::hide()

{

switch (hide\_flag) {

case 1:

shape.setFillColor(Color::Black);

hide\_flag++;

break;

case 2:

shape.setFillColor(get\_color());

hide\_flag--;

break;

}

}

void Square::transform()

{

switch (transform\_flag) {

case 1:

width -= 30;

transform\_flag++;

break;

case 2:

width += 30;

transform\_flag--;

break;

}

shape.setRadius(width);

}

void Square::remove(RenderWindow\* window)

{

window->clear();

}

Square\* Square::clone()

{

Square\* square = new Square(width, color, x, y);

return square;

}

void Square::set\_memento(SquareMemento& memento)

{

color = memento.basic.color;

x = memento.basic.x;

y = memento.basic.y;

width = memento.width;

shape.setRadius(width);

shape.setPointCount(4);

shape.setPosition(x, y);

shape.setFillColor(color);

}

void Square::get\_memento(SquareMemento& memento)

{

memento.basic.color = color;

memento.basic.x = x;

memento.basic.y = y;

memento.width = width;

}

**SquareMemento.cpp:**

#include "SquareMemento.h"

ostream& operator<<(ostream& out, const SquareMemento& memento)

{

out << memento.basic << "\n";

out << memento.width << "\n";

return out;

}

istream& operator>>(istream& in, SquareMemento& memento)

{

in >> memento.basic;

in >> memento.width;

return in;

}

**SquareUtils.cpp:**

#include "SquareUtils.h"

void SquareUtils::save\_state(Square\* square, int counter)

{

ifstream in(filename);

in.seekg(counter, ios::beg);

in >> memento;

square->set\_memento(memento);

in.close();

}

int SquareUtils::load\_state(Square\* square, int before)

{

square->get\_memento(memento);

ofstream out(filename, ios::app);

out << memento;

int after = out.tellp();

int numberOfBytesWritten = after - before;

square->set\_bytes\_written(numberOfBytesWritten);

out.close();

return after;

}

1. **Опис інтерфейсу програми (керівництво користувача) і**

**5. Опис тестових прикладів.**

Після запуску програми користувачу видається вікно, на якому просять натиснути F1 для відкриття меню, а також вікно сцени. Після натискання F1 корстувачу показується меню для взаємодії з програмою:

Натисніть F1 щоб перейти до меню

1: рух фігури

2: додати фігуру

3: видалити фігуру

4: клонувати фігуру

5: додати до композиту

6: змінити колір

7: деформація фігури

8: збереження до файлу

9: завантаження з файлу

ЛКМ: обрати фігуру

ESC: вихід з програми

Після вибору фігури на сцені за допомогою кліку по ній, натиснувши на шостий пункт меню, користувачу запропонують обрати колір для зміни кольору фігури.

При виборі другого пункту меню користувачу запропонують обрати тип фігури, розміри, колір та положення.

При натисканні на восьмий пункт меню сцену буде збережено до файлу, ім’я якого вказане в аргументах командного рядка.

При виборі дев’ятого пункту меню буде завантажено раніше збережений стан сцени з відповідного файлу.

1. **Аналіз помилок (опис усунення зауважень).**